Compsci 201
More Sorting, Backtracking
Part 1 of 4

Susan Rodger
April 3, 2020
T is for …

• Alan Turing, Turing Test, Turing award
  • From WWII to philosophy to math to computing
• Tree
  • From Search to Game to …
Announcements

• APT-6 due Tuesday, April 7

• Assignment P5 Percolation
  • Tell us solo or partner form due yesterday
  • Code due Thursday, April 9

• Discussion 11 Monday, April 6
  • Pre-discussion coming soon

• Exam 2 is April 10
  • Online, flexible hours, extra time, open notes
  • Your own work

• APT Quiz 2 is April 11-15
  • Your own work
If you need an extension for work

- Fill out the form on the course web page, under forms tab
- Then take the extension
- If you need more time, take it again

- Try not to get too far behind!
PfFFiA

- Finish/Review sorting
  - Loop invariants with efficient and inefficient sorts
  - How a priority queue works
  - Bubblesort focus, Oh No!

- Backtracking: NQueens and more
  - Canonical problem-solving and programming
Simple, O(n^2) sorts

- **Selection sort** --- n^2 comparisons, n swaps
  - Find min, swap to front, increment front, repeat

- **Insertion sort** --- n^2 comparisons, no swap, shift
  - *stable*, fast on sorted data, slide into place

- **Bubble sort** --- n^2 everything, slow*
  - Catchy name, but slow and ugly*

  *this isn't everyone's opinion, but it should be

- **Shell sort**: quasi-insertion, fast in practice
  - Not quadratic with some tweaks
Case Study: SelectionSort

- Canonical O(n^2) algorithm/code

```java
void sort(List<T> list) {
    for (int j = 0; j < list.size()-1; j++) {
        int min = j;
        for (int k = j+1; k < list.size(); k++) {
            if (list.get(k).compareTo(list.get(min)) < 0) {
                min = k;
            }
        }
        swap(list, min, j);
    }
}
```
Case Study: SelectionSort

• Invariant: on \( j^{th} \) pass, \([0, j)\) is in final sorted order
• Nested loop re-establishes invariant

```java
public void sort(List<T> list) {
    for (int j = 0; j < list.size() - 1; j++) {
        int min = j;
        for (int k = j + 1; k < list.size(); k++) {
            if (list.get(k).compareTo(list.get(min)) < 0){
                min = k;
            }
        }
        swap(list, min, j);
    }
}
```
Reminder: Loop Invariant

• Statement: true each time loop begins to execute
  • During loop execution it may become false
  • The loop re-establishes the invariant
  • Typically stated in terms of loop index
    • Pictures can help reason about code/solution

• Helps to reason formally and informally about the code you’re writing
  • Can I explain the invariant to someone?
Bubblesort isn't much code

- Swap adjacent elements when out of order
  - From beginning to end, then end-1, end-2, ...
  - After n passes, last n-elements in place

```java
public void sort(List<T> list) {
    for (int j = list.size() - 1; j > 0; j--) {
        for (int k = 0; k < j; k++) {
            if (list.get(k + 1).compareTo(list.get(k)) < 0) {
                swap(list, k, k + 1);
            }
        }
    }
}
```
Timing of $n^2$ and other sorts

<table>
<thead>
<tr>
<th>size</th>
<th>JavaUt</th>
<th>Quicks</th>
<th>MergeS</th>
<th>PQSort</th>
<th>Insert</th>
<th>Select</th>
<th>Bubble</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>0.0242</td>
<td>0.0267</td>
<td>0.0674</td>
<td>0.0182</td>
<td>0.3628</td>
<td>0.4089</td>
<td>0.9051</td>
</tr>
<tr>
<td>20000</td>
<td>0.0184</td>
<td>0.0098</td>
<td>0.0172</td>
<td>0.0184</td>
<td>0.7696</td>
<td>0.9190</td>
<td>2.2390</td>
</tr>
<tr>
<td>30000</td>
<td>0.0218</td>
<td>0.0095</td>
<td>0.0101</td>
<td>0.0136</td>
<td>1.3161</td>
<td>2.1264</td>
<td>5.0625</td>
</tr>
<tr>
<td>40000</td>
<td>0.0125</td>
<td>0.0112</td>
<td>0.0135</td>
<td>0.0122</td>
<td>2.9527</td>
<td>4.0436</td>
<td>9.7758</td>
</tr>
<tr>
<td>50000</td>
<td>0.0170</td>
<td>0.0204</td>
<td>0.0193</td>
<td>0.0212</td>
<td>5.5933</td>
<td>6.5917</td>
<td>16.8241</td>
</tr>
<tr>
<td>60000</td>
<td>0.0240</td>
<td>0.0216</td>
<td>0.0234</td>
<td>0.0188</td>
<td>5.7161</td>
<td>8.8939</td>
<td>25.0218</td>
</tr>
<tr>
<td>70000</td>
<td>0.0235</td>
<td>0.0215</td>
<td>0.0264</td>
<td>0.0222</td>
<td>8.9807</td>
<td>17.1442</td>
<td>31.0544</td>
</tr>
<tr>
<td>80000</td>
<td>0.0299</td>
<td>0.0427</td>
<td>0.0311</td>
<td>0.0247</td>
<td>9.8888</td>
<td>16.3135</td>
<td>43.7167</td>
</tr>
<tr>
<td>90000</td>
<td>0.0305</td>
<td>0.0289</td>
<td>0.0417</td>
<td>0.0269</td>
<td>12.9683</td>
<td>19.6739</td>
<td>48.7305</td>
</tr>
<tr>
<td>100000</td>
<td>0.0278</td>
<td>0.0324</td>
<td>0.0289</td>
<td>0.0322</td>
<td>17.3192</td>
<td>26.9431</td>
<td>66.2827</td>
</tr>
</tbody>
</table>
More efficient $O(n \log n)$ sorts

- Divide and conquer sorts:
  - *Quick sort*: fast in practice, $O(n^2)$ worst case
  - *Merge sort*: stable, fast, extra storage

- Other sorts:
  - Heap sort: priority queue sorting
  - Radix sort: uses digits/characters (no compare)

- $O(n \log n)$ *is optimal for comparing*
  - But, Radix is $O(n)$ ??
Stable, Stability

- **Stable:** respect order of equal keys when sorting
  - First sort by shape, then by color: Stable!
    - Triangle < Square < Circle; Yellow < Green < Red
Merge Sort

• Idea: Divide and Conquer
• Divide list into two halves
• Sort both halves (smaller problem)
• Merge the two sorted halves

9 5 1 4 3 6 2 7
Merge Sort

• Idea: Divide and Conquer
• Divide list into two halves
• Sort both halves (smaller problem)
• Merge the two sorted halves

9 5 1 4 3 6 2 7
9 5 1 4 3 6 2 7
1 4 5 9 2 3 6 7
1 2 3 4 5 6 7 9

divide list into 2 halves
recursively sort each half
merge the two sorted list
What does recursively sort mean?

**Merge Sort**

- Use the same Merge Sort algorithm
  - Divide list into two halves
  - Sort both halves (smaller problem)
  - Merge the two sorted halves

9 5 1 4
What does recursively sort mean?

Merge Sort

• Use the same Merge Sort algorithm
  – Divide list into two halves
  – Sort both halves (smaller problem)
  – Merge the two sorted halves

9 5 1 4

9 5 1 4
divide list into 2 halves

5 9 1 4

5 9 1 4
cursively sort each half

1 4 5 9

1 4 5 9
merge the two sorted list
Merge two sorted lists

- Both lists are sorted.

1 4 5 9

2 3 6 7

Find the smallest from front of two lists
Merge two sorted lists

• Both lists are sorted.

4 5 9

2 3 6 7

1

Find the smallest from front of two lists
Merge two sorted lists

• Both lists are sorted.

\[
\begin{array}{c}
4 & 5 & 9 \\
1 & 2 \\
\end{array}
\quad \begin{array}{c}
3 & 6 & 7 \\
\end{array}
\]

\textbf{Find the smallest from front of two lists}
Merge two sorted lists

• Both lists are sorted.

\[ 4 \ 5 \ 9 \quad 6 \ 7 \]
\[ 1 \ 2 \ 3 \]

*Find the smallest from front of two lists*
Merge two sorted lists

• Both lists are sorted.

5 9 6 7

1 2 3 4

Find the smallest from front of two lists
Merge two sorted lists

- Both lists are sorted.

1 2 3 4 5

9                      6 7

Find the smallest from front of two lists
Merge two sorted lists

- Both lists are sorted.

Find the smallest from front of two lists
Merge two sorted lists

• Both lists are sorted.

1 2 3 4 5 6 7

Find the smallest from front of two lists
Merge two sorted lists

• Both lists are sorted.

Find the smallest from front of two lists
MergeSort idea for code

mergesort(data)
    n = length of data
    if n is 1:
        return data
    else:
        d1 = mergesort(first half of data)
        d2 = mergesort(second half of data)
        return merge(d1, d2)
Time for MergeSort n items: $T(n)$

mergesort(data)
    n = length of data
    if n is 1:
        return data
    else:
        d1 = mergesort(first half of data)
        d2 = mergesort(second half of data)
        return merge(d1, d2)
**Time for MergeSort n items: T(n)**

```plaintext
mergesort(data)

n = length of data

if n is 1:
    return data

else:
    d1 = mergesort(first half of data)  \( T(n/2) \)
    d2 = mergesort(second half of data) \( T(n/2) \)
    return merge(d1, d2) \( O(n) \)

\[ T(n) = 2 \ T(n/2) + O(n) \quad = \quad O(n \log n) \]
```
Quicksort - Idea

• Pivot – select and adjust $< \text{pivot}, \text{pivot}, > \text{pivot}$
  – Select one of the elements
  – Put it where it belongs in sorted order
  – Put elements less than it, to its left
  – Put elements greater than it, to its right
• Recursively sort the elements to its left
• Recursively sort the elements to its right
Quicksort - Idea

• Pivot – select and adjust < pivot, pivot, > pivot
• Recursively sort the elements to its left
• Recursively sort the elements to its right

5 9 1 4 3 6 2 7
Quicksort - Idea

- Pivot – select and adjust < pivot, pivot, > pivot
- Recursively sort the elements to its left
- Recursively sort the elements to its right

Select pivot
Adjust
Recurse left
Result
Recurse right
Result
Quicksort: fast in practice

• Invented in 1962 by Tony Hoare, didn't understand recursion:
  • Canonical $T(n) = 2T(n/2) + O(n)$, but
    • Worst case is $O(n^2)$, bad pivot. Shuffle first?

```java
void doQuick(List<T> list, int first, int last) {
    if (first >= last) return;

    int piv = pivot(list, first, last);
    doQuick(list, first, piv-1);
    doQuick(list, piv+1, last);
}
```

$\begin{array}{c|c|c}
\leq & x & > x \\
\end{array}$

pivot index
Pivot is $O(n)$

- Invariant: $[\text{first}, \text{p}] \leq \text{list.get(first)}$
- Invariant: $(\text{p}, \text{k}) > \text{list.get(first)}$

```java
private int pivot(List<T> list, int first, int last) {
    T piv = list.get(first);
    int p = first;
    for (int k = first + 1; k <= last; k++) {
        if (list.get(k).compareTo(piv) <= 0) {
            p++;
            swap(list, k, p);
        }
    }
    swap(list, p, first);
    return p;
}
```
Stable, $O(n \log n)$ in average and worst, $O(n)$ best!
  - In practice lots of data is "close" to sorted

Invented by Tim Peters for Python, now in Java
  - Replaced merge sort which is also stable

Engineered to be correct, fast, useful in practice
  - Theory and explanation not so simple

https://www.youtube.com/watch?v=NVlJHj-1rT4
Summary of O(n log n) sorts

• Timsort: hybrid of merge and insertion?
  • Fast in real world: Python, Java 7+, Android

• What’s the best O(n log n) sort to call?
  • The one in the library you have access to
    • Arrays.sort or Collections.sort

• Changing how you sort:
  • .compareTo() or .compare()
In computer science, a **sorting** algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order.

Brian Fox

GNU Bash Shell (developer)
First employee at Free Software Foundation
First online banking system at Wells Fargo

There’s nothing that I am better at than everyone else, except being me. There’s no secret to being me. Follow your interests and work hard at them. Then you will play bass better, program better, cook better, ride motorcycles better, or anything else that you really want to do.

https://lifehacker.com/im-brian-fox-author-of-the-bash-shell-and-this-is-how-1820510600
Compsci 201
More Sorting, Backtracking
Part 2 of 4

Susan Rodger
April 3, 2020
How does a Priority Queue work?

• Implemented with a Heap
  • Tree that is stored in an array.
  • It is NOT a tree
  • But easier to think of it as a tree
  • It REALLY is an array
How does a priority queue work?

• Implementation with a heap
  • Tree that is stored in an array
  • Min heap, remove always returns the min
  • Could make it a max heap
    • Use comparator to change
• Root is in index 1
  • Left child: index 2*k
  • Right child: index 2*k + 1
Heap is an array, visualize as Tree

Root at 1
Left child: $2^k$
Right child: $2^k + 1$
Heap is an array, visualize as Tree

Node 10: index 2
Left child: 17
Right child: 25

Node at index k
Left: index 2*k
Right: index 2*k+1
Heap is an array, visualize as Tree

Node 10: index 2
- Left child: index 4
- Right child: index 5

Node at index k
- Left: index 2*k
- Right: index 2*k+1
Heap is an array, visualize as Tree

Node 17: index 4
- Left child:
- Right child:

Node at index k
- Left: index 2\*k
- Right: index 2\*k+1
Heap is an array, visualize as Tree

Node 17: index 4
Left child: index 8
Right child: index 9

Node at index k
Left: index 2*k
Right: index 2*k+1
Properties of a Min-Heap

• Each node is smaller than its children
• Where is the minimal node?
  • At the root of the tree
  • At the front of the array
• A heap is always balanced!
Min-Heap – Where is smallest?
Note each node smaller than children
Remove the minimum element

• Can’t remove the root
• Swap root with last element in array.
• Remove min (now last element in array)
• Fix the heap
Remove the minimum
Remove the minimum

- Min element
Remove the minimum

• Swap min with last
Remove the minimum

• Remove min (last element)
Remove the minimum

- Not a min-heap! Must fix
Remove the minimum

- Compare 25 with lchild and rchild – swap with min
Remove the minimum

- Compare 25 with lchild and rchild – swap with min
Remove the minimum

• Compare 25 with lchild and rchild – swap with min
Remove the minimum

- Compare 25 with lchild and rchild – swap with min
N elements – Time to remove min?

- Swap
- Remove min
- Adjust
N elements – Time to remove min?

- Swap $O(1)$
- Remove min $O(1)$
- Adjust $O(\log n)$

Total: $O(\log n)$
Add element to a min-heap

• Put the element at the end of the array
• Not a heap anymore!
• Bubble element up path to fix heap!
  • Compare element to parent, swap if needed
Add Element to min-Heap

• Add 8
Add Element to min-Heap

• Add 8
Add Element to min-Heap

- Fix the path – compare 8 to parent
Add Element to min-Heap

• Swap 8 and 13
Add Element to min-Heap

• compare 8 and 10
Add Element to min-Heap

• swap 8 and 10
Add Element to min-Heap

- compare 8 and 6 – no need to swap – min-Heap!
N elements – Time to add element?

- Add to array
- Adjust
N elements – Time to add element?

- Add to array \( O(1) \)
- Adjust \( O(\log n) \)

Total: \( O(\log n) \)
Review – Sort with PriorityQueues

• How can we sort N elements using Priority Queue?
  • Add all elements to pq, then remove them
  • Every operation is O(log N), so this sort?
  • O(N log N) – basis for heap sort

```java
void sort(List<T> list) {
    PriorityQueue<T> pq = new PriorityQueue<>(list);
    list.clear();
    while (pq.size() > 0) {
        list.add(pq.remove());
    }
}
```
WOTO – Priority Queues


---

MyLondon

London coronavirus: Ocado selects some special customers to join priority queue

... Ocado selects some special customers to join priority queue ... The queues have sometimes been over 350,000 people long, with reports of ... 6 days ago
Compsci 201
More Sorting, Backtracking
Part 3 of 4

Susan Rodger
April 3, 2020
A Story about Bubble Sort
Prof. Owen Astrachan

- PhD at Duke, Stayed at Duke
- Hates Bubble sort, thinks it is the worst sort ever
- Obsessed with Bubblesort!
Steve Wolfman and Rachel Pottinger, Duke 1997, now Profs at UBC
Wrote a paper

• “Why bubble is not my favorite sort”
• Submitted it to a conference
• It was rejected!
Another paper accepted!

Bubble Sort: An Archaeological Algorithmic Analysis

Owen Astrachan 1
Computer Science Department
Duke University
ola@cs.duke.edu

Abstract

Text books, including books for general audiences, invariably mention bubble sort in discussions of elementary sorting algorithms. We trace the history of bubble sort, its popularity, and its endurance in the face of pedagogical assertions that code and algorithmic examples used in early courses should be of high quality and adhere to established best practices. This paper is more an historical analysis than a philosophical treatise for the exclusion of bubble sort from books and courses. However, sentiments for exclusion are supported by Knuth [17]. “In short, the bubble sort seems to have nothing to recommend it, except a catchy name and the fact that it leads to some interesting theoretical problems.” Although bubble sort may not be a best

1 Introduction

What do students remember from their first programming courses after one, five, and ten years? Most students will take only a few memories of what they have studied. As teachers of these students we should ensure that what they remember will serve them well. More specifically, if students take only a few memories about sorting from a first course what do we want these memories to be? Should the phrase Bubble Sort be the first that springs to mind at the end of a course or several years later? There are compelling reasons for excluding discussion of bubble sort, but many texts continue to include discussion of the algorithm after years of warnings from scientists and educators. For example, in a popular new breadth-first text [6] bubble sort is given
11/08/77

procedure(NUM);

/* SORT sorts BUFFER into alphabetical order
it presently uses bubble sort and an index array */

decl NUM, fixed;
decl (L,J,K) fixed;
dcl WHOLE, fixed;

WHOLE, J = 0;

do I = 0 to NUM - 1 by 1;

if BUFFER(I * 8) + 5 = 0
then WHOLE = WHOLE + 1;
else do:
INDEX(J) = J * 8;
J = J + 1;
end;

end;

INDEX = INDEX - WHOLE
NUM = NUM - WHOLE;

I = NUM;
do while I > 0;

J = 0;
do while J < I - 1;

J = J + 1;
if BINASC(BUFFER(INDEX(J)))\binasc(BUFFER(INDEX(J+1))) >
BINASC(BUFFER(INDEX(J+1)))\binasc(BUFFER(INDEX(J+1)+1))
then do:
K = INDEX(J);
INDEX(J) = INDEX(J + 1);
INDEX(J + 1) = K;
end;

end;

end;
Not needed

Can be tightened considerably
Donald Knuth (Turing 1974)

• “In short, the bubble sort seems to have nothing to recommend it, except a catchy name, and the fact that it leads to some interesting theoretical problems.”
Jim Gray (Turing 1998)

- Bubble sort is a good argument for analyzing algorithm performance. It is a perfectly correct algorithm. But it's performance is among the worst imaginable. So, it crisply shows the difference between correct algorithms and good algorithms.
Feah. I love bubble sort, and I grow weary of people who have nothing better to do than to preach about it. Universities are good places to keep such people, so that they don't scare the general public.

(continued)
Brian Reid (Hopper 1982)

I am quite capable of squaring N with or without a calculator, and I know how long my sorts will bubble. I can type every form of bubble sort into a text editor from memory. If I am writing some quick code and I need a sort quick, as opposed to a quick sort, I just type in the bubble sort as if it were a statement. I'm done with it before I could look up the data type of the third argument to the quicksort library.

I have a dual-processor 1.2 GHz Powermac and it sneers at your N squared for most interesting values of N. And my source code is smaller than yours.

Brian Reid

who keeps all of his bubbles sorted anyhow.
I have read your article and share your view that Bubble Sort has hardly any merits. I think that it is so often mentioned, because it illustrates quite well the principle of sorting by exchanging.

I think BS is popular, because it fits well into a systematic development of sorting algorithms. But it plays no role in actual applications.

Quite in contrast to C, also without merit (and its derivative Java), among programming codes.
Obama on Sorting
When he was a senator running for President
What is the most efficient way to sort a million 32-bit integers?
Although bubble sort is one of the simplest sorting algorithms to understand and implement, its $O(n^2)$ complexity means that its efficiency decreases dramatically on lists of more than a small number of elements. Even among simple $O(n^2)$ sorting algorithms, algorithms like insertion sort are usually considerably more efficient.

Due to its simplicity, bubble sort is often used to introduce the concept of an algorithm, or a sorting algorithm, to introductory computer science students. However, some researchers such as Owen Astrachan have gone to great lengths to disparage bubble sort and its continued popularity in computer science education, recommending that it no longer even be taught.\[^4\]
Musical Bubblesort
Compsci 201
More Sorting, Backtracking
Part 4 of 4

Susan Rodger
April 3, 2020
Backtracking and Blob Fill

- Explore a move (blob fill) if it works? Fabulous!
  - If it does not work? Undo the move, try again

- Similar to Sudoku solving? Crossword puzzles?
  - Tentatively try number of word, follow through
  - May need to undo and try alternative
Exhaustive Search

• Can explore every possible move in tic-tac-toe
  • Cannot explore every possible move in chess
• Brute-force doesn't work
  • Be smart, try move? Then undo, try another

• Backtracking in search tree
  • Smart pruning
N-Queens: Know History

• Can we place N queens on N×N board so no queen attacks another
  • 4×4 or 8×8 or …
8 Queens – What fun!

```java
public Queens(int n){
    mySize = n;
    myBoard = new QBoardGUI(n);
    if (solve(0)){
        myBoard.print();
    }
}
```

```java
public static void main(String[] args){
    int size = 8;
    double start = System.nanoTime();
    Queens q = new Queens(size);
    double end = System.nanoTime();
    System.out.printf("time: %fn",(end-start)/1e9);
}
```
Nqueen Concepts

• For each column c in [0..N)
  • Try to place queen in each row of column c
  • grid[r][c] ok? Place queen, try c+1
    • If not ok? Or Doesn't work? Try next row, r+1

• When have all queens been placed?
  • If c == N and success? Done!
  • Can't do column c? return false, c-1 continues
Code for Nqueens Backtracking

[https://coursework.cs.duke.edu/201spring20/backtracking-sp20/](https://coursework.cs.duke.edu/201spring20/backtracking-sp20/)

- Done when \( c == N \)
  - Place queens, recurse, unplace and try again
- Return true
  - All placed
  - Recursive
- Use myBoard
  - Track moves

```java
public boolean solve(int col){
    if (col == mySize) {
        return true;
    }
    // try each row until all are tried
    for(int r=0; r < mySize; r++){
        if (myBoard.safeToPlace(r,col)){
            myBoard.setQueen(r,col,true);
            if (solve(col+1)){
                return true;
            }
            myBoard.setQueen(r,col,false);
        }
    }
    return false;
}
```
Backtracking Summary

• Enumerate all possible moves/choices
  • Nqueen? Each column and each row in column
  • Blob-fill? Each neighbor: fill, and unfill

• Board often two-dimensional array/grid
  • Record move, recurse, undo if not done
Backtracking APTs

• Often use `grid[][]` to store state/moves
  • In Java this is actually an array of arrays

• `int[][] a = new int[4][4]` for example
  • What is `a[0]`? What is `a[0][0]`?

• Often move must be explicitly undone
  • Sometimes just try everything
Backtracking APTs

• Often use `grid[][]` to store state/moves
  • In Java this is actually an array of arrays

• `int[][] a = new int[4][4]` for example
  • What is `a[0]`? What is `a[0][0]`?

  It is a row! It is the top left element!

• Often move must be explicitly undone
  • Sometimes just try everything
Collaborative APT Solving

- [https://www2.cs.duke.edu/csed/newapt/gridgame.html](https://www2.cs.duke.edu/csed/newapt/gridgame.html)
- Shall we play a game?
  - Each player plays perfectly
- If I go here, will you win?

Grid Game APT

Problem Statement

In a simple game, two players take turns placing 'X's in a 4x4 grid. Players may place 'X's in any available location ('.') in the input) that is not horizontally or vertically adjacent to another 'X'. The player who places the last 'X' wins the game. It is your turn and you want to know how many of the moves you could make guarantee you will win the game, assuming you play perfectly.
All-seeing and All-knowing

• Given a board, how many winning moves?
  • Can't tell, can determine all possible moves
  • Make a move, ask if it's a winner
    • Can't tell? Make move and repeat
What can we do with a board?

• Can you determine if \([r][c]\) is legal?
  • \([1][0]\) is legal, why?
  • \([3][1]\) is NOT legal, why?

• Suppose there are no legal moves? Answer: Zero/0
• Suppose I place an 'X' and then ask
  • How many ways to win does opponent have?
  • If answer is Zero/0, what does placing 'X' do?

• This leads to backtracking, believe the code!!!
APT GridGame Ideas

• Have boolean method `winWithBoard(board)`
  • For each legal move
    • Place on board – board now changed
    • Call `winWithBoard(board)` -- if true? I lose
    • Undo move --- take off board and continue

• If you could see ahead, you'd know if the move was good. `winWithBoard` is an oracle
What Do We Believe?

• Predictions by oracles ...
All-seeing and All-knowing

- Given a board, how many winning moves?
  - Can't tell, can determine all possible moves
  - Make a move, ask if it's a winner
    - Can't tell? Make move and repeat
All-seeing and All-knowing

• Given a board, are there any moves possible?
  • If no moves possible … last move was winner!
    • Last person to place an X wins

• So, try every X, with each one …
  • Ask recursively if winner
    • Make move and ask …
      – Make move and ask …
private int winCount(char[][] board) {
    int wins = 0;
    for(int r=0; r < 4; r++){
        for(int c=0; c < 4; c++){
            if (canMove(board,r,c)) {
                board[r][c] = 'X';
                int opponentWins = winCount(board);
                if (opponentWins == 0) {
                    wins += 1;
                }
                board[r][c] = '．';
            }
        }
    }
    return wins;
}
Red to try each open space and ... 

```
" .X.X"
"..X."
".X.."
"...."
```

```
" .X.X"
"X .X"
".X.."
"...."
```

```
"X .X"
".X X"
".X ."
"...."
```

```
"X .X"
"..X.
".X X"
"..X"
```

```
".X.X"
"..X."
"X.."
"...."
```

```
".X.X"
"X X." 
".X ."
"...X"
```

```
"X X"
"..X."
"X.."
"...."
```

```
"X .X"
"..X.
"X.."
"...X"
```

Red loses

Red wins!

Make a move, ask oracle (recursion) how many winners after move? 
Oracle tries all moves and says none ... meaning made move winner
Red to try each open space and ...

`.X.X`
"..X."
".X.."
"...."

`.X.X`
"X.X."`
".X.."
"...."

`.X.X`
".X.X"
".X.X"
"...."

`.X.X`
".X.X"
".X.X"
"X X"

Make a move, ask oracle (recursion) how many winners after move?

Red loses

Red wins!

Oracle tries all moves and says none ... meaning made move winner
private int winCount(char[][] board) {
    int wins = 0;
    for(int r=0; r < 4; r++){
        for(int c=0; c < 4; c++){
            if (canMove(board,r,c)){
                board[r][c] = 'X';
                int opponentWins = winCount(board);
                if (opponentWins == 0){
                    wins += 1;
                }
                board[r][c] = '.';
            }
        }
    }
    return wins;
}
Don't Know Much about History

- Usenet, Chess, Checkers, …
  - Alan Biermann, Tom Truscott: Internet Pioneer
WOTO on Backtracking

Donald Knuth (Turing 1974)

- Writing multiple volumes “The Art of Computer Programming”
- Title is “Professor of The Art of Computer Programming” at Stanford

- Pipe Organ in his home with 812 pipes
- Played the organ in the Duke Chapel!